Multiple Integer Divisions with an Invariant Dividend and Monotonically Increasing or Decreasing Divisors

Daisuke Takahashi(✉)

Center for Computational Sciences, University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki 305-8577, Japan
daisuke@cs.tsukuba.ac.jp

Abstract. In this paper, we propose an algorithm for multiple integer divisions with an invariant dividend and monotonically increasing or decreasing divisors. In such multiple integer divisions, we show that if the dividend and divisors satisfy a certain condition, then if only one quotient is calculated by division first, the remaining quotients can be obtained by correcting the previously calculated quotients at most once. The proposed algorithm is up to approximately 1.90 and 1.85 times faster than the 64-bit unsigned integer division instruction of the Intel 64 architecture and Intel Short Vector Math Library (SVML) on the Intel Xeon Platinum 8368 processor, respectively.

Keywords: Multiple integer divisions · invariant dividend · approximate quotient

1 Introduction

Integer division is a widely used arithmetic operation, being included in many applications. In general, division is known to be slower than addition, subtraction, and multiplication. For this reason, algorithms have been proposed to perform division using only addition, subtraction, and multiplication. In particular, algorithms for division by invariant integers [3,5,8–10] and algorithms for integer division using reciprocals [1,13,14] have been proposed. Additionally, 32- and 64-bit integer division using double-precision floating-point arithmetic has been proposed [11].

In this paper, we consider multiple integer divisions when the dividend is invariant and the multiple divisors are monotonically increasing or decreasing. Such multiple integer divisions appear when converting an integer to Montgomery representations [12] for multiple moduli [15].

The remainder of this paper is organized as follows. Section 2 presents the proposed algorithm for multiple integer divisions with an invariant dividend and monotonically increasing or decreasing divisors. Section 3 presents the performance results. Finally, Sect. 4 presents concluding remarks.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
O. Gervasi et al. (Eds.): ICCSA 2023, LNCS 13957, pp. 393–401, 2023.
https://doi.org/10.1007/978-3-031-36808-0_26

2 Proposed Algorithm

An unsigned integer division is defined by the quotient $q = \lfloor a/b \rfloor$ and the remainder $r = a - bq$ $(0 \le r < b)$, where the dividend is a and the divisor is b.

We consider multiple integer divisions when the dividend is invariant, and the multiple divisors are monotonically increasing or decreasing. In such multiple integer divisions, we show that if the dividend and divisors satisfy a certain condition, then if only one quotient is calculated by division first, the remaining quotients can be obtained by correcting the previously calculated quotients at most once.

Theorem 1. *Consider integers $a \ge 0$, $b > 0$, and $c \ge 0$. If $ca \le b(b + c)$, then*

$$\left\lfloor \frac{a}{b} \right\rfloor < \left\lfloor \frac{a}{b+c} \right\rfloor + 2. \tag{1}$$

Proof. For any real number x, $x - 1 < \lfloor x \rfloor \le x$ holds. Thus, we have

$$\frac{a}{b} - 1 < \left\lfloor \frac{a}{b} \right\rfloor \le \frac{a}{b}, \tag{2}$$

$$\frac{a}{b+c} + 1 < \left\lfloor \frac{a}{b+c} \right\rfloor + 2 \le \frac{a}{b+c} + 2. \tag{3}$$

From inequalities (2) and (3), the condition $a/b \le a/(b + c) + 1$ for inequality (1) to hold can be obtained as $ca \le b(b + c)$. □

From Theorem 1, if a, b, and c satisfy the condition $ca \le b(b + c)$, the quotient $\lfloor a/b \rfloor$ can be obtained by making at most one correction to the quotient $\lfloor a/(b + c) \rfloor$. Thus, Theorem 1 is applicable when the divisors are monotonically decreasing. When the divisors are monotonically increasing, the quotient can be obtained in the same way by reversing the order of the divisors.

Algorithm 1 shows the proposed algorithm for multiple integer divisions with an invariant dividend and monotonically increasing divisors. If it is known in advance that the condition of Theorem 1 is satisfied for the dividend a, the minimum divisor b_0, and the maximum difference c between adjacent divisors, then multiple integer divisions can be efficiently calculated using Algorithm 1.

The first and second lines of Algorithm 1 calculate the quotient q_{n-1} and remainder r_{n-1}, respectively, for the dividend a and the divisor b_{n-1}. In line 4, the previously calculated quotient q_{i+1} is used as the approximate quotient. In line 5, the remainder r_i is obtained from the dividend a, the divisor b_i, and the quotient q_i. If the remainder r_i is greater than or equal to the divisor b_i, then lines 7 and 8 correct the quotient q_i and remainder r_i.

Algorithm 2 shows the proposed algorithm for multiple integer divisions with an invariant dividend and monotonically decreasing divisors. If it is known in advance that the condition of Theorem 1 is satisfied for the dividend a, the

Algorithm 1. Multiple integer divisions with an invariant dividend and monotonically increasing divisors

Input: $a \geq 0,\ b_0, b_1, \ldots, b_{n-1} > 0,\ b_i \leq b_{i+1}$ for $0 \leq i \leq n-2,\ ca \leq b_0(b_0 + c)$, where $c = \max(b_{i+1} - b_i)$ for $0 \leq i \leq n-2$
Output: $q_i = \lfloor a/b_i \rfloor$ for $0 \leq i \leq n-1$, $r_i = a \bmod b_i$ for $0 \leq i \leq n-1$
1: $q_{n-1} \leftarrow \lfloor a/b_{n-1} \rfloor$
2: $r_{n-1} \leftarrow a - b_{n-1}q_{n-1}$
3: **for** i **from** $n-2$ **downto** 0 **do**
4: $\quad q_i \leftarrow q_{i+1}$
5: $\quad r_i \leftarrow a - b_i q_i$
6: $\quad$ **if** $r_i \geq b_i$ **then**
7: $\quad\quad q_i \leftarrow q_i + 1$
8: $\quad\quad r_i \leftarrow r_i - b_i$.

Algorithm 2. Multiple integer divisions with an invariant dividend and monotonically decreasing divisors

Input: $a \geq 0,\ b_0, b_1, \ldots, b_{n-1} > 0,\ b_i \geq b_{i+1}$ for $0 \leq i \leq n-2,\ ca \leq b_{n-1}(b_{n-1} + c)$, where $c = \max(b_i - b_{i+1})$ for $0 \leq i \leq n-2$
Output: $q_i = \lfloor a/b_i \rfloor$ for $0 \leq i \leq n-1$, $r_i = a \bmod b_i$ for $0 \leq i \leq n-1$
1: $q_0 \leftarrow \lfloor a/b_0 \rfloor$
2: $r_0 \leftarrow a - b_0 q_0$
3: **for** i **from** 1 **to** $n-1$ **do**
4: $\quad q_i \leftarrow q_{i-1}$
5: $\quad r_i \leftarrow a - b_i q_i$
6: $\quad$ **if** $r_i \geq b_i$ **then**
7: $\quad\quad q_i \leftarrow q_i + 1$
8: $\quad\quad r_i \leftarrow r_i - b_i$.

minimum divisor b_{n-1}, and the maximum difference c between adjacent divisors, then multiple integer divisions can be efficiently calculated using Algorithm 2.

The first and second lines of Algorithm 2 calculate the quotient q_0 and remainder r_0, respectively, for the dividend a and the divisor b_0. In line 4, the previously calculated quotient q_{i-1} is used as the approximate quotient. In line 5, the remainder r_i is obtained from the dividend a, the divisor b_i, and the quotient q_i. If the remainder r_i is greater than or equal to the divisor b_i, then lines 7 and 8 correct the quotient q_i and remainder r_i. In Algorithms 1 and 2, when the divisors $b_0, b_1, \ldots, b_{n-1}$ are an arithmetic progression, then c is equal to the absolute value of the common difference.

In integer division, it is sometimes desirable to calculate only the quotient or remainder [2]. In such cases, Algorithms 1 and 2 can be modified to output only the desired result. Figures 1 and 2 show multiple 64-bit unsigned integer divisions with an invariant dividend and monotonically increasing and decreasing divisors, respectively.

We now consider vectorization of Algorithms 1 and 2. In Algorithms 1 and 2, the quotients are obtained by correcting only the previously calculated quotient and this process cannot be vectorized due to data dependency. However,

```
void divinc64(uint64_t *q, uint64_t *r, uint64_t a, uint64_t *b, int n)
/*  Compute q[0:n] = floor(a / b[0:n]) and r[0:n] = a mod b[0:n].
    Requires a >= 0, b[0:n] > 0, b[i] <= b[i + 1] for 0 <= i <= n - 2
    and c * a <= b[0] * (b[0] + c),
    where c = max(b[i + 1] - b[i]) for 0 <= i <= n - 2. */
{
  int i;

  q[n - 1] = a / b[n - 1];
  r[n - 1] = a - b[n - 1] * q[n - 1];

  for (i = n - 2; i >= 0; i--) {
    q[i] = q[i + 1];
    r[i] = a - b[i] * q[i];
    if (r[i] >= b[i]) {
      q[i]++;
      r[i] -= b[i];
    }
  }
}
```

Fig. 1. Multiple 64-bit unsigned integer divisions with an invariant dividend and monotonically increasing divisors

```
void divdec64(uint64_t *q, uint64_t *r, uint64_t a, uint64_t *b, int n)
/*  Compute q[0:n] = floor(a / b[0:n]) and r[0:n] = a mod b[0:n].
    Requires a >= 0, b[0:n] > 0, b[i] >= b[i + 1] for 0 <= i <= n - 2
    and c * a <= b[n - 1] * (b[n - 1] + c),
    where c = max(b[i] - b[i + 1]) for 0 <= i <= n - 2. */
{
  int i;

  q[0] = a / b[0];
  r[0] = a - b[0] * q[0];

  for (i = 1; i < n; i++) {
    q[i] = q[i - 1];
    r[i] = a - b[i] * q[i];
    if (r[i] >= b[i]) {
      q[i]++;
      r[i] -= b[i];
    }
  }
}
```

Fig. 2. Multiple 64-bit unsigned integer divisions with an invariant dividend and monotonically decreasing divisors

by simultaneously correcting multiple previously calculated quotients, this data dependency can be resolved, making it possible to vectorize the process. To do this, the previously calculated quotients for the divisors at multiples of the vector length are used as the approximate quotients. Thus, the value of c in Algorithms 1 and 2 is the maximum difference between divisors at intervals of the vector length.

```
void vdivinc64(uint64_t *q, uint64_t *r, uint64_t a, uint64_t *b, int n)
/*  Compute q[0:n] = floor(a / b[0:n]) and r[0:n] = a mod b[0:n].
    Requires a >= 0, b[0:n] > 0, b[i] <= b[i + 1] for 0 <= i <= n - 2
    and c * a <= b[0] * (b[0] + c),
    where c = max(b[i + VLEN] - b[i]) for 0 <= i <= n - VLEN - 1. */
{
  int i, j;

  for (i = n - 1; i >= n - VLEN; i--) {
    q[i] = a / b[i];
    r[i] = a - b[i] * q[i];
  }

  for (j = n - VLEN - 1; j >= 0; j -= VLEN) {
    for (i = j; i >= j - VLEN + 1; i--) {
      q[i] = q[i + VLEN];
      r[i] = a - b[i] * q[i];
      if (r[i] >= b[i]) {
        q[i]++;
        r[i] -= b[i];
      }
    }
  }
}
```

Fig. 3. Vectorized multiple 64-bit unsigned integer divisions with an invariant dividend and monotonically increasing divisors

Figures 3 and 4 show vectorized multiple integer divisions with an invariant dividend and monotonically increasing and decreasing divisors, respectively. In these programs, the inner loop of the doubly nested loop can be vectorized. These programs assume that the number of divisors n is divisible by the vector length `VLEN`. If n is not divisible by `VLEN`, a remainder loop needs to be executed.

```
void vdivdec64(uint64_t *q, uint64_t *r, uint64_t a, uint64_t *b, int n)
/*  Compute q[0:n] = floor(a / b[0:n]) and r[0:n] = a mod b[0:n].
    Requires a >= 0, b[0:n] > 0, b[i] >= b[i + 1] for 0 <= i <= n - 2
    and c * a <= b[n - 1] * (b[n - 1] + c),
    where c = max(b[i] - b[i + VLEN]) for 0 <= i <= n - VLEN - 1. */
{
  int i, j;

  for (i = 0; i < VLEN; i++) {
    q[i] = a / b[i];
    r[i] = a - b[i] * q[i];
  }

  for (j = VLEN; j < n; j += VLEN) {
    for (i = j; i < j + VLEN; i++) {
      q[i] = q[i - VLEN];
      r[i] = a - b[i] * q[i];
      if (r[i] >= b[i]) {
        q[i]++;
        r[i] -= b[i];
      }
    }
  }
}
```

Fig. 4. Vectorized multiple 64-bit unsigned integer divisions with an invariant dividend and monotonically decreasing divisors

3 Performance Results

For performance evaluation, we compared the execution times of multiple 64-bit unsigned integer divisions using the following four implementations:

1. Implementation using the 64-bit unsigned integer division instruction `div` of the Intel 64 architecture
2. Implementation of the proposed algorithm using the Intel 64 architecture instructions
3. Implementation using the `_mm512_div_epu64()` intrinsic function, which divides packed 64-bit unsigned integers in the Intel Short Vector Math Library (SVML) [6]
4. Implementation of the proposed algorithm using the Intel Advanced Vector Extensions 512 (AVX-512) instructions [7]

The evaluation used the following settings. The batch size n of 64-bit unsigned integer divisions varied from 128 to 1024. In these cases, the divisors, the quotients, and the remainders fit into the L1 cache. The dividend a is set to be a random number in the range $[0, 2^{64}-1]$. For the monotonically increasing divisors b_i $(0 \le i \le n-1)$, b_0 is set to $\lfloor\sqrt{\texttt{VLEN} \times a}\rfloor + 1$, and $b_i = b_0 + i$. For the monotonically decreasing divisors b_i $(0 \le i \le n-1)$, b_{n-1} is set to $\lfloor\sqrt{\texttt{VLEN} \times a}\rfloor + 1$,

and $b_{n-i-1} = b_{n-1} + i$, where the vector length `VLEN` is set to 8. Note that the dividend and divisors satisfy the condition of Theorem 1.

Each batch of 64-bit unsigned integer divisions was executed one million times. The number of 64-bit unsigned integer divisions per second (Mops) was calculated based on the average elapsed time.

The specifications of the platform are shown in Table 1. The Intel Xeon Platinum 8368 processor has 38 cores. However, we evaluated the performance on a single core and a single thread to focus on vectorization. The Intel C compiler (version 19.1.3.304) was used. The compiler options were `icc -O3 -xCORE-AVX512 -qopt-zmm-usage=high`. The compiler option `-O3` specifies to optimize for maximum speed and enable more aggressive optimizations. The compiler option `-xCORE-AVX512` specifies to optimize for Intel processors that support Intel AVX-512 instructions. The compiler option `-qopt-zmm-usage=high` specifies to generate zmm code without restrictions.

Table 2 shows the performance of multiple 64-bit unsigned integer divisions with an invariant dividend and monotonically increasing divisors. When the batch size n is 1024, the proposed algorithm is approximately 1.49 and 1.76 times faster than the 64-bit unsigned integer division instruction of the Intel 64 architecture and Intel SVML on the Intel Xeon Platinum 8368 processor, respectively. The Ice Lake microarchitecture of the Intel Xeon Platinum 8368 processor requires significantly fewer cycles for the Intel 64 architecture integer division instruction `div` than previous Intel microarchitectures such as the Skylake microarchitecture [4], and so the difference in performance between the proposed algorithm and the 64-bit unsigned integer division instruction of the Intel 64 architecture is smaller on the Intel Xeon Platinum 8368 processor.

Table 1. Specifications of the platform

Platform	Intel Xeon Platinum processor
Number of cores	38
Number of threads	76
CPU type	Intel Xeon Platinum 8368 Ice Lake 2.4 GHz
L1 cache (per core)	I-cache: 32 KB D-cache: 48 KB
L2 cache (per core)	1.25 MB
L3 cache	57 MB
Main memory	DDR4-3200 256 GB
Theoretical peak performance	2.918 TFlops
OS	Linux 4.18.0-305.25.1.el8_4.x86_64

Table 2. Performance of multiple 64-bit unsigned integer divisions with an invariant dividend and monotonically increasing divisors (Mops)

Batch size n	Intel 64	Proposed (Intel 64)	Intel SVML	Proposed (Intel AVX-512)
128	336.622	507.979	600.133	1110.869
256	338.892	503.765	604.713	1069.585
512	339.121	505.398	603.699	1068.396
1024	338.809	505.617	604.131	1065.383

Table 3. Performance of multiple 64-bit unsigned integer divisions with an invariant dividend and monotonically decreasing divisors (Mops)

Batch size n	Intel 64	Proposed (Intel 64)	Intel SVML	Proposed (Intel AVX-512)
128	336.906	640.423	718.866	949.816
256	338.859	638.441	717.021	941.146
512	338.772	637.106	713.086	936.785
1024	338.666	637.954	703.699	933.340

Table 3 shows the performance of multiple 64-bit unsigned integer divisions with an invariant dividend and monotonically decreasing divisors. When the batch size n is 1024, the proposed algorithm is approximately 1.88 and 1.33 times faster than the 64-bit unsigned integer division instruction of the Intel 64 architecture and Intel SVML on the Intel Xeon Platinum 8368 processor, respectively.

4 Conclusion

In this paper, we proposed an algorithm for multiple integer divisions with an invariant dividend and monotonically increasing or decreasing divisors. In such multiple integer divisions, we showed that if the dividend and divisors satisfy a certain condition, then if only one quotient is calculated by division first, the remaining quotients can be obtained by correcting the previously calculated quotients at most once. Performance results showed that the proposed algorithm is faster than the 64-bit unsigned integer division instruction of the Intel 64 architecture and Intel SVML.

Acknowledgments. This work was supported by JSPS KAKENHI Grant Number JP22K12045.

References

1. Alverson, R.: Integer division using reciprocals. In: Proceedings of 10th IEEE Symposium on Computer Arithmetic (ARITH 1991), pp. 186–190 (1991)
2. Brent, R.P., Zimmermann, P.: Modern Computer Arithmetic. Cambridge University Press, Cambridge (2010)
3. Drane, T., Cheung, W.C., Constantinides, G.: Correctly rounded constant integer division via multiply-add. In: Proceedings of 2012 IEEE International Symposium on Circuits and Systems (ISCAS 2012), pp. 1243–1246 (2012)
4. Fog, A.: Instruction tables: lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD, and VIA CPUs (2022). https://www.agner.org/optimize/instruction_tables.pdf
5. Granlund, T., Montgomery, P.L.: Division by invariant integers using multiplication. In: Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 1994), pp. 61–72 (1994)
6. Intel Corporation: Intel C++ compiler classic developer guide and reference (2021). https://www.intel.com/content/dam/develop/external/us/en/documents/cpp_compiler_classic.pdf
7. Intel Corporation: Intel 64 and IA-32 architectures software developer's manual, volume 1: Basic architecture (2022). https://cdrdv2-public.intel.com/671436/253665-sdm-vol-1.pdf
8. Jacobsohn, D.H.: A combinatoric division algorithm for fixed-integer divisors. IEEE Trans. Comput. **C-22**, 608–610 (1973)
9. Lemire, D., Bartlett, C., Kaser, O.: Integer division by constants: optimal bounds. Heliyon **7**, e07442 (2021)
10. Möller, N., Granlund, T.: Improved division by invariant integers. IEEE Trans. Comput. **60**, 165–175 (2011)
11. Monniaux, D., Pain, A.: Formally verified 32- and 64-bit integer division using double-precision floating-point arithmetic. In: Proceedings of 29th IEEE Symposium on Computer Arithmetic (ARITH 2022), pp. 128–132 (2022)
12. Montgomery, P.L.: Modular multiplication without trial division. Math. Comput. **44**, 519–521 (1985)
13. Robison, A.D.: N-bit unsigned division via n-bit multiply-add. In: Proceedings of 17th IEEE Symposium on Computer Arithmetic (ARITH 2005), pp. 131–139 (2005)
14. Rodeheffer, T.L.: Software integer division (2008). https://www.microsoft.com/en-us/research/wp-content/uploads/2008/08/tr-2008-141.pdf
15. Takahashi, D.: Computation of the 100 quadrillionth hexadecimal digit of π on a cluster of Intel Xeon Phi processors. Parallel Comput. **75**, 1–10 (2018)